Guidelines for a Long Term Competitive Intrusion Detection System

Erwan Lemonnier - *r1@defcom.com*

5th October 2001

This thesis work was performed at Defcom Sweden (Apelbergsgatan 60, Box 1093, 10139 Stockholm) from January to june 2001, in cooperation with the Royal Institute of Technology (KTH) of Stockholm.

I would like to thank especially Olav Bandman and Mads Dam who supervised this project on KTH side , and Lovisa Haraldsson and all the members of OnGuard's development team from Defcom, with whom it was and still is a pleasure to work. Special regards also go to Guillaume Girard for his wise and sharp critic.

Contents

Abstract

Intrusion Detection Systems (IDSs) are the computer equivalent of office burglar alarms: they aim at monitoring computer networks for detecting attacks and intrusions. IDSs are becoming one of the main security components in secured network environments. Though rewarding, their mission is also challenging and IDSs are facing a few major obstacles. Analyzing these obstacles in order to define the guidelines for an IDS that would remain efficient on a long term scale is the project of the present report.

The study starts with an overview of practical network security, followed by a review of current IDS technologies. This provides us with a ground for identifying the main challenges facing IDSs, as for example monitoring encrypted or gigabit traffic, improving alert-flow relevancy and resisting to evasion technics. A first approach to solve some of these challenges consists in designing efficient alert filters, which we will illustrate. A second approach is to consider IDSs from a different point of view, seeing them as information flow processing systems. It thus appears that an efficient IDS could be built by integrating multiple IDSs in a common alert processing structure.

Introduction

Computer security has become a head of the news topic. While computer systems are invading every corner of our lives and getting complex and hard to secure, the knowledge of how to exploit their vulnerabilities is simultaneously reaching more and more users. Intrusion Detection Systems (IDSs) are playing an increasingly important role as a security component to build secured network environments. After about 15 years of development, IDSs are now starting to be available on a large scale. Yet, there remain a few obstacles that current IDS technologies are still unable to address. The near future of IDSs will mainly depend on their ability to find solutions to these challenges.

Hence the interest of looking in detail at these issues, in order to identify some requirements that the next generation of IDSs should follow to remain efficient. The purpose of this essay is to analyze these challenges and define some guidelines for building the next generation of IDS. We will then suggest an architecture for such an IDS.

The first part consists in an overview of practical computer security, in order to understand the context in which IDSs are used. We will then study the technologies currently used in common IDSs, while discussing their relative efficiency. Once the inner-working of IDSs explained, we will focus on the challenges that IDSs are facing, in order to identify some requirements for a long term competitive IDS and make suggestions on how to implement it.

Chapter 1

Introduction to practical network security

Computers tend nowadays to be organized in large networks connected to Internet, and are used in an increasing number of sensitive applications, such as online banking and shopping, control of hardware installations and public infrastructures. Meanwhile, as more and more critical tasks are delegated to computers, computer security becomes a matter of first importance, hence the interest of understanding what everyday practical computer security consists of, which is the purpose of this chapter. But let us first look at a more theoretical point of view.

1.1 Theoretical Computer Security

Computer security threats are usually classified in three categories referred to as *Confidentiality*, *Integrity* and *Availability* (CIA). *Confidentiality* concerns the protection of data from unauthorized disclosure, *Integrity* deals with unauthorized modification of data, and *Availability* with giving a reliable access to data.

Confidentiality, Integrity and Availability are implemented in a computer system[1] through a so-called 'security model'. From a theoretical point of view, every operation in a computer system, can be seen as a 'subject' accessing an 'object'. Both subject and object can be any kind of computer entity such as a process, a file or a hardware device, depending on the context. The part of the computer system in charge of managing these accesses is called a Reference Monitor. The Reference Monitor allows access based on a sequence of access control rules. For these rules to be effective, the computer system should also provide the Reference Monitor with a reliable method to identify and authenticate subjects and objects. The way all these tasks are implemented defines a security model. Figure 1.1 shows a simplified Reference Monitor at work[2].

[1] 'computer system' refers to any structure involving one single computer to many interconnected computers.

[2] Figure coming from *'Intrusion Detection'*, by Terry Escamilla, ISBN 0471290009.

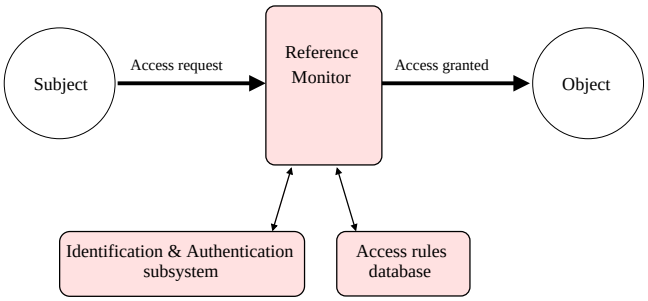

Figure 1.1: A reference monitor regulates subjects accessing objects

A theoretical definition of computer attacks could hence be 'any activity aiming at breaking a security model'. Interconnecting computers in networks opens the door to a wide panel of attacks against computer systems.

1.2 Network architecture

To understand security in computer networks, one should start with understanding the structure and components of typical computer networks.

1.2.1 Definitions

1.2.1.1 Computer & Network

The word 'computer' may refer to many different hardware configurations. The most widespread architecture is the usual PC, while other architectures such as diskless workstations or mainframes are usually found in laboratory environments. Computers may be interconnected to form a network, enabling a permanent data link between computers. Many technologies are available to interconnect computers: frame relay, Ethernet, token ring or wireless. In large networks, specialized devices are required to handle communications between smaller network segments. Such devices are bridges, hubs, switches and routers.

1.2.1.2 Protocol

Once a reliable physical link is established between computers, data may be exchanged. Yet, since the link in itself is nothing more than an electrical cable, protocols have to be defined to enable reliable data exchange between computers, and a special architecture needs to be created to route data between computers.

These protocols are often organized in layers, going from the physical link to the program application layer. Each of these layers contains protocols in charge of a particular aspect of the data exchange: transferring data on a local segment, identifying each computer in a large network, enabling a reliable data exchange despite data losses, etc. The most widespread network protocol stack is the TCP/IP protocol suite which is

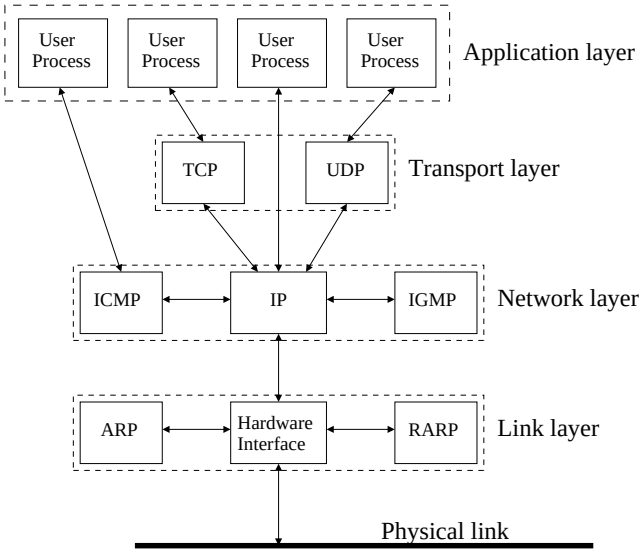

Figure 1.2: The TCP/IP protocol suite

used for Internet. It is made of 4 layers: link, network, transport and application layer. The main protocols used in the TCP/IP stack are showed on figure 1.2[3].

1.2.1.3 Network applications

As soon as some reliable mean of exchanging binary information is available to interconnected computers, doors are opening to an incredibly rich panel of applications. An endless list of services is now available on computer networks: world wide web, email, file sharing, online databases, streamed music and video, interfaces to other systems (telephone, hardware...), etc. Any program running on a computer connected to a network has the possibility to send and receive data from other computers. Most network applications are designed according to the client-server model in which a program called 'server' is running on a computer (also called server, by extension) and listening for incoming data sent by a 'client'. Figure 1.3 shows a typical stateful client-server connection.

1.2.2 The DMZ model

There is a potentially infinite number of possible network structures, due to the variety of methods available to build and interconnect platforms and networks. Understanding a given network's structure is an important issue while thinking about its security, and we will therefore need in the rest of this document a reference model of common network architecture. The standard reference model in computer security is the

[3]Figure coming from *'TCP/IP illustrated vol.1'*, by W.R. Stevens, ISBN 0201633469.

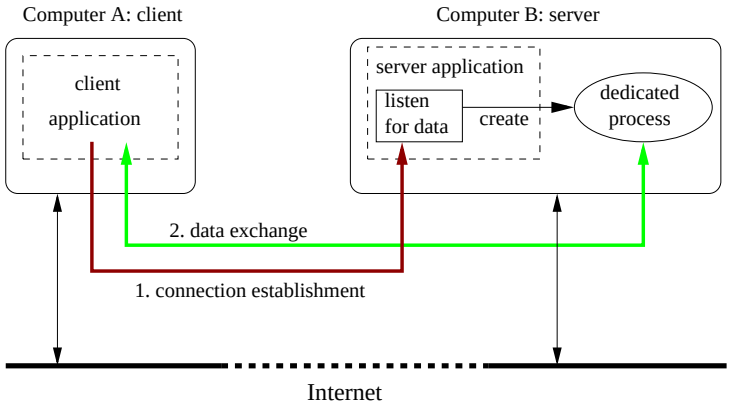

Figure 1.3: A stateful client-server connection

DMZ (*'DeMilitariZed'*) shown in figure 1.4[4]. This kind of network can be divided in 3 areas: the Internet, the Internal Network, and the DMZ, all interconnected through a router, which almost always also acts as a firewall (see §1.4). The DMZ usually contains servers, such as HTTP, mail or FTP servers, that need to be accessible from both the Internal Network and the Internet, and thus require lower access restrictions from the Internet than computers located in the Internal Network (which justifies the term 'demilitarized'). The router/firewall will consequently be configured differently for traffic going between the Internet and the DMZ, and between the Internet and the Internal Network, being more restrictive for the second. The Internal Network usually contains the company's employees' personal workstations, as well as some internal file or database servers that should not be accessed from the Internet.

The DMZ model, although very simplified compared to real life network architectures, is still accurate enough to illustrate most of the existing security threats, as well as the role of the most common security components, as described in the next sections.

1.3 Security threats in computer networks

1.3.1 Theoretical analysis of network weaknesses

Computer networks have inherent weaknesses that expose them to security threats. These weaknesses can be listed as following[5]:

Sharing: Network resources (file system, devices, application servers...) are designed to be accessible from any point in a network, and are thus inherently exposed.

System complexity: A large network may interconnect different operating systems, use different network protocols and offer different services. Moreover, it may

[4]Figure inspired by *'Hacking Exposed'*, ISBN 0072127481.
[5]inspired by *'Security in computing'* by C.P. Pfleeger, ISBN 0131857940.

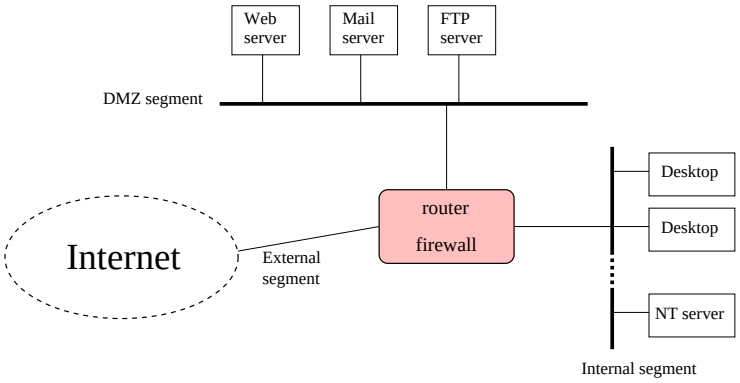

Figure 1.4: The DMZ model

be administrated by different people, with different needs for security, different physical location or different languages. All this put together makes managing and securing a large network a challenging task.

Unknown perimeter: Considering the complexity of large networks, it gets sometimes hard to identify clear network boundaries as well as responsibilities for their administration.

Many point of attacks: Data traveling through a network can be accessed from many different locations. As well, networks may have multiple entry points, with inequal security levels.

Anonymity: Networks in themselves do not provide any mean of authentication. These have to be implemented in the protocols used. Intruders may thus benefit of partial or complete anonymity, or be able to mascarade themselves.

Unknown path: Data can often take many different paths to join two end points in a network. Network components, moreover, can not keep track of all data passing through them during more than a short time. Tracing some network activity back to its sources may thus be technically challenging, not to mention problems related to multiple geographical locations, laws and languages.

1.3.2 Common network attacks

Here is a non-exhaustive list of some common type of attacks that can be used in a network environment:

Buffer overflow: Comes in two flavors: in a remote buffer overflow, data formated in a special way and sent to a server application will cause the server to crash (Denial of Service) or to execute some code hidden in the data, thus allowing to gain access to information, remote shell, passwords, etc. In a local buffer overflow, a user already logged on a computer uses a breach in a local application to force it to execute code in an attempt to gain higher privilege.

Denial of Service (DoS): Network data packets or program arguments formated in a certain way may cause the receiving application (server application, system or user program...) to stop, consume extensive amount of CPU or even crash its underlying operating system.

Sniffing: An intruder spies the network traffic passing through a given network point, thus being able to catch information, non encrypted logins and passwords, etc.

Brute forcing: Some servers require identification before opening a connection, but are designed in such a way that systematic login attempts are possible, until a valid login is found

Bad configuration: Server applications or firewalls are not correctly configured and let some door open to intrusion.

Social engineering: This technics consists into fooling a computer user into revealing sensitive information allowing an intruder to get further access into a network, usually by pretending being part of the network administration staff.

1.3.3 Practical causes for network vulnerabilities

There are many more possible attacks than those listed in §1.3.2, and a real attack will usually combine many of them. The main practical causes for these vulnerabilities can be identified as:

- **Weak security model**. Protocols and applications are often designed without considering security as a main issue. Some examples resulting of weak security models are clear text protocols that can be sniffed, or bad password policy enabling password guessing, or trusted relations between computers, between which no authentication is required.

- **Inherent theoretical breach**. When a protocol or an algorithm has defects in its design itself. Examples of this are hijacking for the TCP protocol, or flawed cryptographic algorithms, such as WEP (Wireless Encryption Protocol).

- **Implementation breach**. An algorithm or protocol has been incorrectly implemented, thus presenting security vulnerabilities. Buffer overflows result from such programming mistakes, and are usually due to a lack of control on received arguments.

- **Configuration breach**. When an application, device or a system, even a secured one, is improperly configured, thus giving its user some higher privilege than required.

- **Lack of security awareness**. Computer users are usually unaware of what the security risks are. They will use weak passwords, install unsafe applications, and be vulnerable to social engineering.

1.3.4 Network target points

Attackers will mainly focus on 4 kind of targets in a network:

1. Server programs that are listening for connections. They can be located on network servers, workstations or network structural components (router, switch...). They may suffer from buffer overflow or denial of service vulnerabilities, be badly configured or allow password brute forcing.

2. Network protocols. Clear text protocols that can be sniffed, protocols enabling brute forcing or hijacking...

3. Physical devices. An attacker (also called 'insider') having physical access to a sensitive network component may gain control of it if it has weak of flawed user identification mechanisms.

4. Computer users. Experience has shown that social engineering works very well.

Several security components are focusing on reducing the risk related to these four weak points.

1.4 Security components in a computer network

The process of building a secured computer network can be described as a small set of successive stages. The first step is to clearly define the level of security that needs to be reached, and design a corresponding security model. The second step is to implement this security model, by analyzing the network components used (hardware, operating systems, software applications, protocols...) and choosing appropriate security components to implement the security model. A knowledge of each component's security model and vulnerabilities is needed.

Many security components are available to build or enforce security in a network. Below is a description of the most common ones:

Secured Operating System: an operating system which architecture is implementing a strong security model. Since every operating system actually implements some form of security model, it gets important to understand it and be able to compare it with other security models' implementations. For example, Unix is a reasonably secured operating system, providing user and process identification and authentication, delimiting file and memory access rights. Yet, kerberos over unix provides a higher level of security by implementing stronger authentication means. Experience shows that using well secured operating systems is a very important first step in the process of securing a computer network. Without this prerequisite, a high level of security won't be reached.

Firewalls: a firewall is responsible for delimiting areas in computer networks and applying rules on how network packets circulate from one area to an other. The firewall stands at the crosspoint between these areas. On a network layer level,

it controls packet forwarding between network areas, based on configurable access rules. On a higher application level, it may control user sessions, do content filtering, etc.

Intrusion Detection Systems (IDS): IDSs are the computer equivalent of burglar alarms and are further described in § 1.5.

Vulnerability scanner: vulnerability scanners are automated tool scanning a network for known vulnerabilities that may be used by intruders against network components. A vulnerability scanner is built around a database of vulnerabilities which needs to be regularly updated in order to include newer vulnerabilities. This process is challenging considering the rate at which new vulnerabilities are discovered and vulnerability scanners are therefore never totally up to date. They are useful both for network administrators to regularly audit their network security and for potential intruders to find possible targets.

Virus scanners: virus scanners aim at detecting 3 categories of dangerous program files circulating on computer networks: viruses, worms and Trojan horses. A virus is a self reproducing program that is encapsulated in a host application which serves as a mean of transport, thus propagating itself from computers to computers as the host application is itself replicated. A worm is like a virus, but does not need a host application to provide transport. It uses instead some breaches in system applications to forward itself to an other system and run there. Both viruses and worms, beside their replicating functionalities, may also attempt to violate the security model of a system by deleting files, spying on information, etc. Trojan horses, finally, are legitimate programs that have been modified to include some dangerous functionality, like those listed previously. Virus scanners will usually be located on workstations or network gateways to filter the content of received files.

Cryptography: cryptography may be used at different levels of a security model. It may be used to provide strong identification and authentication (certificates, kerberos, PKI architecture...), as well as to simply protect data while transfered over network protocols (ssl, https...) or while saved on host file systems, thus increasing confidentiality. The algorithms used, the quality of their software implementation and the key length used are the main factors determining the strength of a cryptographic application.

In the DMZ model, these components could be theoretically used to secure the network in the following way:

- Secured operating systems would be used on all servers. Since the level of security required for desktops is lower, they may run less secured and more user-friendly operating systems.

- A firewall would run on the router. Using firewalls is nowadays a minimal prerequisite.

- Three network based IDSs would monitor respectively the DMZ segment, the Internal segment and External segment. Host based IDSs would run on the servers.
- Virus scanners would run on every desktop computer and on servers through which file transfer may occur (mail and ftp servers). A virus scanner functionality may be included in the firewall running on the router.
- The network administrator would run an up to date vulnerability scanner on a regular basis, and apply security patches to operating systems and server applications throughout the network.
- As much as possible of the data circulating on the Internal segment, and between the Internal segment and the DMZ would be encrypted (email via imap/ssl, ssh instead of telnet, https instead of http, etc.). Sensitive data would be encrypted before being stored.

There are many other security components than those listed above. Practical computer security is dealing with how to combine these components in order to give a computer system the level of security required for the activity it is supporting, thus implementing a given security model.

1.5 Role of Intrusion Detection Systems in a secured network

1.5.1 Definition of an IDS

Intrusion Detection Systems (IDSs) are monitoring programs aiming at detecting 'intruders' who are acting 'illegally' in a computer system. The signification of 'intruder' and 'illegal' depends on the limits set by a security model, but can be generally defined as a person or process performing unauthorized operations on a network element (server, router, workstation...) in an attempt to gain more control over it.

There are two main categories of IDSs: host based and network based. A host based IDS is running as a process on a host computer and monitors sensitive activities on this computer, such as unauthorized access or modification of files. A network based IDS consists in a sniffer program listening to the network traffic in an attempt to detect suspicious activity over network protocols. Both host and network based IDSs will generate alarms when detecting suspicious activity. These alarms should trigger a response from the network administrator or some automated response tool. A host based IDS will usually be located on critical computers such as servers, while a network based IDS should be located at strategical points in a network, in order to have access to relevant traffic.

1.5.2 Role and limitations of IDSs

IDSs are the computer equivalent of usual office burglar alarms. They consequently have a passive role in network security. Most of the time, the IDS will raise an alarm

when the attack is already finished, which makes them quite inefficient for blocking ongoing attacks. But they can help at detecting and understanding the attack, in order to efficiently repair its consequences and prevent other similar attacks.

The technics used to build IDSs, their advantages and their limitations will be described in depth in the rest of this document.

Chapter 2

A review of IDS technologies

This chapter is focusing on enumerating the technics commonly used in different parts of IDSs. Special care will be given to technics related to efficient IDS filter design, on which the author has been specifically working.

2.1 Standard IDS architecture

An IDS, whether it is host or network based, usually consists in 4 distinct successive layers: a *sensor*, running some *filters* that are generating an *alert-flow* directed to a *monitoring central*, as illustrated on figure 2.1. This general model can be used to describe host based IDSs, as well as network based IDSs or even Distributed IDSs (DIDSs).

The sensor is responsible for gathering relevant data from a monitored system. The filters are small specialized programs parsing the data provided by a sensor in order to detect possible attack patterns. Suspect patterns trigger the filters into sending alerts. The alerts produced by all the filters are gathered into an alert-flow. This alert-flow is then directed to a monitoring central, that it will reach after being filtered and pre-processed. On the monitoring central, an end-user monitors and interprets the alert-flow.

This layered model is of interest to us since it successfully describes the most common IDS architectures. But alternative models also exist: some IDS are using more simple models, and some others are based on completely different architectures, inspired for example by the human immune system or by multi-agent technologies.

2.2 Distributed IDS architecture

The previous model can be extended to Distributed IDSs, in which sensors, filters, alert-flow processing components and monitoring centrals can be located on different systems. A common alert-flow is then created by gathering alerts from multiple groups of sensors and filters. The previous 4 layer model can still describe such distributed architectures, as shown on figure 2.2.

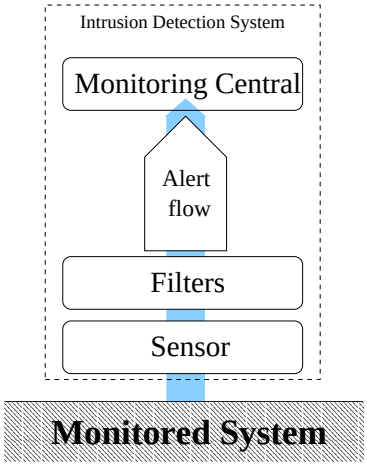

Figure 2.1: A 4 layer model for standard IDS architecture.

In the rest of this essay, the term IDS may refer without distinction to IDS and DIDS.

2.3 Sensors

The information gathered by a sensor should be relevant for finding attack signs. Since attacks depend on the target system, the sensor will have a different structure and gather different information depending on the system it is monitoring.

2.3.1 Host based sensors

A host based sensor is monitoring an operating system and its processes. It is typically a program running on a monitored computer, and checking for access to critical files, changes in access rules, process activity, login activity, etc.

2.3.2 Network based sensor

A network based sensor monitors the network traffic on a network segment, and is quite similar to a network sniffer. It consists mainly of a network adapter put in promiscuous mode[1], a TCP/IP stack implementation, and some programming interface to this stack, providing the filters with relevant traffic information.

[1]When in promiscuous mode, a network card receives all the traffic passing through its local network segment, not only the packets directed to this card.

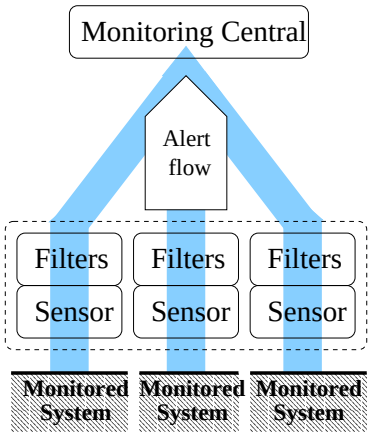

Figure 2.2: Distributed IDS architecture.

2.3.3 Switch based sensor

Using network based sensors in switched network environment implies to have one sensor listening on each switched segment. In order to reduce the number of sensors required, an alternative is to design a switch based sensor, located on the switch and having access to all traffic passing through the switch. Such sensors are currently under development and still not available.

2.3.4 Choosing sensors' types and locations

Choosing the appropriate combination of sensors and their location is an important issue which requires an analysis of the monitored network's structure, its security level and its vulnerable points.

Host based sensors should be located on relevant computers such as highly exposed servers or servers requiring a higher level of security. Network or interface based sensors should monitor relevant traffics and could hence be located on each segment connected to a firewall, on each of the monitored network's entry points from the Internet or on segments where critical servers are running.

2.4 Filter technology

The efficiency of an IDS highly depends on the quality of its filters, since they are doing the actual intrusion detection work. The design and configuration of these filters is a crucial issue when building an IDS.

2.4.1 Attack recognition strategies

There are a few ways for a filter to detect attack signs inside the data stream provided by the sensor.

2.4.1.1 Signature filter

Signature filters are the simplest and most widespread type of filter. A signature filter basically looks for one specific signature in the data flow provided by a sensor. An attack usually possesses some kind of signature that identifies it. This signature often consists in one or more specific binary pattern found in a given file or network packet. The signature can be described as a boolean relation called rule.

Although simple to implement, signature filters have a number of limitations. They are able to recognize an attack only when they 'know' a signature for this attack, and thus require continuous updates of their signature database as well as continuous research work to analyse new attacks and find their signatures. Moreover, a slight change in the attack scenario may be enough to alter the attack signature and thus fool a signature filter, which is consequently vulnerable to technics such as polymorphic attacks (see § 3.5).

An other limitation is that the process of discovering signatures is complex and not reliable. It relies on an individual having both detailed access to attack logs and the technical skills to analyse them, or who gets to learn how a new attack works. This person should then inform competent authorities (such as securityfocus) of his findings. These authorities then propagate the information among security professionals including those in charge of updating IDSs filters and signatures. Hence, signature filters will not detect attacks until they have been widely discovered, and probably widely used.

2.4.1.2 Protocol Anomaly filter

The term 'anomaly filter' is often used to refer to two different kind of filters defined here as *protocol anomaly* and *statistical anomaly* filters. We shall here define the first category.

Theory

A protocol anomaly filter is a filter looking for protocol misusage. 'Protocol' should here be interpreted as any official set of rules describing the interaction between elements in a computer system. A protocol anomaly filter is consequently designed to analyze specifically one protocol, and requires a model of this protocol's 'normal' usage. Protocols always have a theoretical usage, corresponding to their official description in documents such as RFCs[2], but experience shows that they are seldom implemented and used in complete accordance with these official descriptions. A model for a protocols 'normal' usage can thus be defined as the superposition of the official and practical area of usage of this protocol. Any use of this protocol outside these intersected areas

[2]Request For Comments are the official written definitions of the protocols and policies of the Internet.

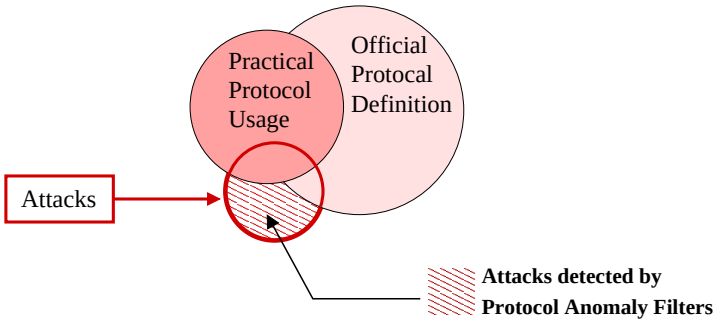

Figure 2.3: Attacks detected by Protocol Anomaly Filters.

can be considered as a protocol anomaly. Figure 2.3 illustrates this situation through the two overlapping colored circles representing practical and official protocol usage.

The interesting point is that experience shows that the majority of the attacks can be considered as protocol usage anomalies. The reason lays in the fact that most of the attacks are exploiting breaches in badly defined areas of protocols, in which special cases have been neglected in the protocol standard itself as well as in its implementations.

Protocol anomaly filters are thus able to detect all attacks that are using protocols outside of their normal usage area, which includes in particular new attacks that may not yet have been registered by computer security authorities, as shown on figure 2.3. This ability of detecting new attacks, added to the fact that they don't require signature database updates and have the same comparatively long lifetime as the protocol they are monitoring, asserts the superiority of protocol anomaly filters over signature filters.

Practice

When running above a network based sensor, protocol anomaly filters would disassemble the data packets for each network protocol, and check if they are built in compliance with the protocol standards, as described in RFCs or equivalents. The common implementations of this protocols should also be studied, in order to define the limits of what is officially and unofficially the standard for this protocol.

An example of protocol anomaly is to give an extra large username (many kilobytes) with the USER command of the FTP protocol while logging onto a FTP server. Though extra large arguments are not often considered in official protocol standards as explicitly illicit, they should be considered in practice as a suspicious sign, since they are not justified by normal usage and may be an attempt to exploit buffer overflows or denial of service. More examples of protocol anomalies are given in chapter 4.

2.4.1.3 Statistical Anomaly filter

A statistical anomaly filter is designed to monitor a system and detect statistically exceptional events. The idea is to build a model of a system's normal behavior under safe conditions, and then to periodically compare the system's behavior against this model.

If the difference exceeds some limit, an alert is generated. Protocol anomaly filter can be considered as a special case of statistical anomaly filter applied to a protocol syntax.

In a network based context, the behavior monitored could be a user's activity over the network, or a protocol traffic. The model for a normal behavior would be obtained by sampling the corresponding behavior under a period of time representative of safe activity. The same parameters would then be sampled again on a regular basis, and the result compared with the reference model. In a host based context, the behavior monitored would be more focused on user activities, program activities, file and device accesses, etc.

How evolutive a reference model should be is a source of dilemma. Indeed, systems behaviors tend to change with time, and statistical anomaly filters should consequently be designed to allow regular updates of their reference model. On the other hand, if this model is updated too often, an intruder could spread his activity over a period of time long enough to let the filter 'learn' his behavior and accept it in its model as a normal behavior.

2.4.1.4 Neural networks

An other approach to attack detection is to use neural networks to detect attack patterns inside a stream of system information. The idea is to select representative information streams coming from sensors and to connect them to a neural network. The neural network is then 'taught' to recognize specific attack patterns while the attacks are simulated on the monitored system.

While elegant in their transparency and flexibility, neural networks present the drawback of not being accurate in pattern recognition: they can hardly ever offer 100% detection accuracy, and thus should not be used as a main filtering technic in an IDS. They can on the other hand be used as a parallel tool helping to confirm attack diagnosis.

2.4.2 Alert generation

An other issue in filter design is the generation of alert messages to acknowledge the anomaly or attack detected. This can be divided into two areas: defining a standard for an alert format, and increasing alert accuracy through internal filter programming technics.

2.4.2.1 Alert format

Defining an efficient format for alerts is required in order to give accurate information to the human end-user of the IDS as well as to enable alert manipulation by intermediate programs managing the alert-flow.

There is currently no standard format for describing alerts, despite the many suggested candidates. Typical solutions involve using general description languages such as XML, or defining proprietary format based for example on key-value pairs. A difficulty is to define a standard flexible enough to be able to describe new attacks, but structured enough to enable automated analysis of its messages. The IETF is about to

release a RFC describing a format called IDWG that is believed to become the alert standard.

2.4.2.2 False-positives

With a theoretically perfect filter, each alert sent would exactly correspond to a real attack. In practice, a filter can often miss an attack or on the other hand send an alert while no attack is perpetrated. An alert of this second category is called false-positive and occurs typically when a filter interprets some legitimate activity as revealing an attack. Reducing the amount of false-positives is one of the main issue in IDS design.

2.4.2.3 Filter design for alert-flow control

There are two criteria that a filter should match to provide efficient alert generation:

1. Avoid false-positives
2. One attack should not trigger a flood of alerts

These two measures aim at reducing the alert-flow and increasing alert relevancy. These requirements can be met by using filter components processing the alert-flow itself. Such a component can be added inside the filter, and works as follow: when the attack detection component of the filter generates an alert, it stores it temporarily in an alert log which is parsed and emptied periodically by an alert filter component, as shown on figure 2.4. This component analyses the alert log and can:

- simply forward relevant alerts
- build a macro-alert summarizing multiple related alerts
- apply alert level filtering rules to implement a higher level of attack analysis

Accurate alert generation is related to the more general problem of alert-flow processing, which will be deeper studied in the rest of this document in §3.2 and §5.4.3.

2.4.3 Filter configuration

A filter needs to be configured to specifically fit the system it is monitoring.

From a general point of view, the more information a filter possesses concerning the system it is monitoring, the more accurate its analysis can get. This is especially true with anomaly filters that require to be tightly adapted to each specific system's behavior in order both to be effective and to avoid sending too many false-positive alerts. On the other hand, computer systems are evolving environments which requires filter configurations to be regularly updated to fit system changes. Otherwise, the filters will start generating an increasing flow of false-positives. Filter configuration presents therefore a dilemma: the tighter the configuration fits the network, the more efficient the filter, but the less adaptable it gets to system changes. An excess in either direction results in a loss of quality of the alert-flow (false-positives).

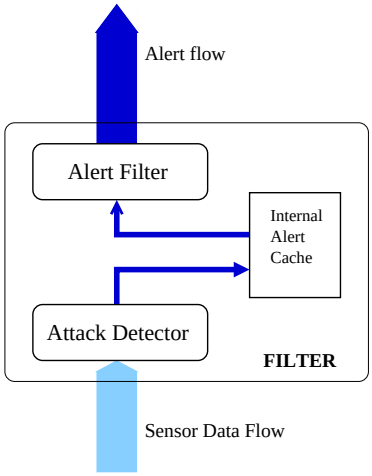

Figure 2.4: Filter's internal alert processing component

Special care is therefore required to design highly configurable filters that can be accurately adapted to a specific system. Their configuration should be easy to do and modify, if possible through appropriate user interfaces. Besides, if filters require periodic manual configuration, they might create a huge work load.

An interesting but not yet explored technique to override this problem of periodic filter configuration is to use self configurable filters. The idea is to design the filters in such a way that they can learn from the system they are monitoring the information that they require to analyse it efficiently. This would especially apply to anomaly filters. Yet, it is likely that some information would still have to be given manually.

2.5 Alert-flow

IDS filters generate an alert-flow which is ultimately directed to an end-user, whether it is an individual or an automated response tool. For this end-user, monitoring and interpreting the alert-flow is a complex task that can be made simpler by inserting alert-flow preprocessing components between the filters and the end-user. Most actual IDSs offer no or very simple alert-flow preprocessing, and it is therefore a promising research field.

2.5.1 User interface

In the simplest case of IDS, alerts are just stored in a log file and it is up to the end-user to analyse it properly.

To facilitate this task for the human end-user, most modern IDSs provide a (Graphical) User Interface to the alert-flow. The design of such an interface is an important

issue, since its purpose is to help the end-user to analyse the raw alert-flow generated by the IDS. An in-depth analysis of IDS GUIs' requirements is beyond the scope of this document, but we can list a few common features that they should possess:

- They should be designed in order to clearly show important alert information, such as IP addresses and alert type and severity.
- They should help in sorting alert by severity.
- They should provide information on what the alert means.
- They should possess some customizable filtering capabilities, to filter unrelevant alerts.

2.5.2 Alert storage

All alerts generated by the IDS should be stored permanently for later inspection in a dedicated database, in order to facilitate post-attack analysis and forensics. A database is preferable to a raw log file, since it is more adapted to automated report generation and query work. According to the first security requirement for IDSs exposed in § 2.8.1, this database should be secured, preferably located on a dedicated computer, isolated from the monitored network and connected to the IDS through a separate channel. Alerts could be stored encrypted.

2.5.3 Alert-flow processing

Dynamically increasing the relevancy of the alert-flow between the filters and the end-user is one of the key issue on the way to designing more efficient IDS. This is a challenging problem, closely related to artificial intelligence. Ultimately, a goal is to include an expert system inside IDSs in order to assist the end-user in identifying attacks.

We saw in §2.4.2.3 that alert level filtering can be done inside the filter itself. A second layer of alert-flow processing can be done beyond the point where the alert-flows generated by each of the filters get merged. Different kind of processing are then possible to reduce the alert-flow and increase its relevancy. They will be further studied in §5.4.3, but we can already define 2 categories:

Dynamic Processing Dynamic processing is obtained by modifying dynamically the alert-flow before it reaches the end-user. The dynamic processing engine should not introduce too much delay in forwarding the alert-flow and the quality of its analysis is thus limited by time and delivery requirements.

Passive Processing Passive processing is a form of data-mining obtained by analyzing the alert database, in order to identify attack patterns some time after the related alerts were generated. It is not limited by time as for dynamic processing, and can thus provide a deeper and more accurate analysis of alert logs. It could theoretically be able to detect attacks spread over a longer time scale or distributed attacks. On the other hand, due to the delay between an attack and its recognition, this technique would not be of use for stopping ongoing attacks.

2.6 IDS administration

An IDS's administration is a heavy task, especially with distributed or remotely managed IDS. Specific administration frameworks are required to support at least the following 4 activities:

IDS Monitoring An IDS should provide self monitoring capabilities, in order to detect the failure of some IDS components and allow to automatically or manually fix it. This can be enhanced through a framework for automatically probing the IDS status.

Configuration IDS configuration is a complex and time consuming task. As previously shown in § 2.4.3 for filters, an appropriate configuration framework is strongly advised, in particular for distributed IDS.

Updating As for configuration, having an updating framework to automatically or remotely upgrade various components of an IDS is also an issue, especially in distributed or remotely managed IDS where having a person doing upgrades manually on site can be hard or expensive.

Alert Investigation Being able to efficiently query the alert log is another issue when starting to investigate past sequence of events, hence the importance of having a proper database to store alerts.

2.7 Beyond detection

What happens once the alerts have reached the monitoring end-user ?

2.7.1 Response protocol

In the case of remotely managed IDS, the monitoring staff may not have direct physical access to the relevant components of a monitored system. They may not even be in the same country, or know the way the network administration staff of the monitored system is working. Therefore, a pre-establish response protocol is required, which defines the measures to take accordingly to the nature of alerts.

2.7.2 Self-defense and shunning

Although IDSs are passive monitoring components, they can be changed into dynamic defensive components by using a technics called shunning. Shunning is used to dynamically disconnect a monitored system from the source host or network of a detected attack. The problem with shunning lays in the difficulty of reliably identifying the source of an attack.

Shunning is performed by reconfiguring on the fly and remotely the filtering rules of a packet forwarding device controlling the traffic between the monitored network and the outside. It usually consists in blocking for a given amount of time all traffic between the monitored network and the source IP address of the attack. Yet, modifying

network routing rules based on suspicious IP sources is a risk since IP does not provide reliable identification: IP packets can be forged with spoofed or random source addresses. Using shunning thus opens the door to a wide variety of attacks based on IP spoofing and which can result on a denial of service on the monitored system. An attacker could for example force a monitored network into getting disconnected from a crucial server by spoofing attack traffic coming from this server.

Shunning is moreover in opposition with the second security requirement exposed in §2.8.1: if an IDS is compromised, it should not compromise the monitored system. Implementing such a rule basically implies that the IDS should not have any way of acting on the monitored system, and thus perform shunning.

2.7.3 Report generation

Storing the alerts in a database offers us an interesting option: periodical report generation. Such a report can be of use for long term surveys, for providing the monitored system administration staff with regular snapshots of the security related activities on their system. Such information will help in administrative decisions concerning system evolution.

2.8 Security consideration for distributed IDS

2.8.1 Security requirements

There are a few theoretical security requirements that IDSs, and especially distributed IDSs, should meet to be considered reliable:

1. IDS components as well as the data they exchange should not be accessible to unauthorized users.

2. If a component of an IDS gets compromised, it should not compromise any other part of the IDS.

3. If the IDS is compromised, it should not compromise the security of the monitored system.

The term compromise here refers to any possible loss of availability, confidentiality or trustability, due for example to internal failures or manipulation by an unauthorized person. In the case of components internal failures, an automated framework for detecting and restarting these components is required. However, detecting a breach of security in an IDS component is hard, and focus should therefore be given into limiting the scope of such a security breach. Hence the former requirements.

2.8.2 Open failure problem

A security component, such as an IDS or a firewall, can usually be classified in 2 categories, depending on whether it provides open or close failure. A component provides

close failure when, upon stopping, it leaves the system it was securing at least as secured as while it was running.

IDSs are passive monitoring components, and they belong as such to the open failure category, which means that when stopping, an IDS leaves its monitored system without protection. Though advised by the third security requirement listed above, the safety of this behavior can be discussed: some IDSs can be vulnerable to DoS attacks and crashed remotely by an attacker, thus leaving the system unprotected. An alternative is to use shunning in order to automatically isolate the monitored network whenever its IDS appears to be down. However, shunning possesses some drawbacks as seen in § 2.7.2. The IDS stability and resistance to DoS attacks is therefore crucial, and the IDS should provide a mechanism for automatically restarting its components if they crash and for alerting end-users of its inner state.

2.8.3 Data exchange and alert protection

The data generated and exchanged by the components of an IDS are a source of information on the monitored system. It is especially true for alerts which may inform on system vulnerabilities. Protecting inter IDS components communications is therefore required, in particular in distributed IDSs configurations for which different components are running on multiple interconnected computers.

A first step into implementing this requirement is to encrypt all data transfers between IDS components. A second step is to physically isolate their communication channels from all untrusted network. This can be done by interconnecting IDS components through a dedicated and isolated LAN. This approach should be preferred to Virtual Private Networks which still rely on the (unproved) security of an underlying operating system. Besides, IDS components should avoid trusting each other but use authentication methods, to avoid spoofing attacks.

Finally, care should be given into protecting the alerts when stored in a log or database. Their access should be controlled through identification and authentication procedures, and they should be stored encrypted.

2.8.4 Stealth IDS

Practice shows that nowadays IDSs are vulnerable to a variety of attacks ranging from IDS evasion techniques to Denials of Service. Each IDS has its own vulnerabilities, and an attacker who knows if a target system is protected by an IDS and which one may be able to build an attack that will bypass the IDS.

It is therefore advised, as a protective measure for the IDS itself, that the IDS remains as stealthy as possible. This is easier to implement with network based sensors: their network interface put in promiscuous mode should not send any data. Yet, even if the sensor does not send data on this interface, there are techniques to detect cards in promiscuous mode by forcing the card to send packets, using ARP queries. A solution consists in modifying the interface on a hardware level to forbid data emission. Being stealthy is on the other hand much harder for host based sensors. Yet it is still possible to partly hide the IDS presence on the host.

Chapter 3

IDS issues and limitations

This chapter focuses on analyzing the limitations and challenges awaiting IDS now and in the next few years.

3.1 Theoretical limitations

3.1.1 Fundamental differences in systems' behaviors

Each computer system connected to a network possesses its own state and its own set of rules defining the way it passes from one state to an other. When monitoring and interpreting some activity, an IDS should know exactly the state and the behavioral rules of the monitored system(s), in order to accurately predict the system's reaction. In practice, the IDS is generally forced to make assumption on what the state is, and to assume that the behavioral rules of the end-system are implementing standard protocols in the same way as the IDS itself. These two assumptions are fundamentally flawed.

This is especially true with network based IDS monitoring the data traffic between two end-systems and making assumptions on how they generate and handle this data exchange. Presently, each operating system has its own implementation of the TCP/IP protocol stack. The official description (RFCs) of most of these protocols, TCP and IP in particular, let some aspects of the protocol implementation up to the programmer. This, in addition to the inherent complexity of these protocols, makes that two different implementations of the TCP/IP stack usually presents differences in the way they handle unusual cases of protocol usage (such as with IP fragment reassembly). Some specific network traffic might be interpreted differently depending on the stack implementation of the system receiving the traffic. Typically, some network packets may be accepted by a system and discarded by an other.

3.1.2 Insertion and Evasion

Differences in TCP/IP stacks implementations of network based IDS and monitored end-systems are opening the door to two attacks called insertion and evasion, that were

described as follow in '*Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*' (T.H. Ptacek & T.N. Newsham, 1998).

'An IDS can accept a packet that an end-system rejects. An IDS that does this makes the mistake of believing that the end-system has accepted and processed the packet when it actually hasn't. An attacker can exploit this condition by sending packets to an end-system that it will reject, but that the IDS will think are valid. In doing this, the attacker is "inserting" data into the IDS. No other system on the network cares about the bad packets.'

On the other hand:

'An end-system can accept a packet that an IDS rejects. An IDS that mistakenly rejects such a packet misses its contents entirely. This condition can also be exploited, this time by slipping crucial information past the IDS in packets that the IDS is too strict about processing. These packets are "evading" the scrutiny of the IDS.'

Network based IDS can be fooled using these two technics. In practice, such technics are called for IDS evasion technics, and are most often based on IP fragmentation, packet dis-ordering or invalid packet forging.

3.1.3 Traffic normalizers

Traffic normalizers are an interesting solution against most insertion and evasion technics. They can be seen as a filter dynamically altering a network traffic in order to force it to follow a safe subset of protocol usage. They act as a proxy for traffic forwarding: they take in entry normal traffic, reassemble it, then regenerate an equivalent but safer traffic and forward it. Traffic normalizers would protect a monitored network from being vulnerable to IDS evasion technics based on differences in TCP/IP stack implementation.

Unfortunately, traffic normalizers are and may stay purely theoretical. It is indeed difficult to define a complete subset of protocol usage that would be free of insertion-evasion possibilities. Moreover, as expressed in issue 38-5 of Phrack: *'until Cisco implement this technology in IOS or Checkpoint do likewise with FW-1, etc., both unlikely prospects in the short to medium term, the implication is that this suite of NIDS*[1] *subversion techniques will to call into question the reliability of NIDS.*'. Traffic normalizers only normalize network traffic, and would still not remove the theoretical need of knowing other system states.

3.2 Alert-flow control challenges

The IDSs ability to efficiently manage the alert-flow is going to become the most important issue in IDS design in the next few years.

3.2.1 False-positive alerts

False-positives are caused by the filters interpreting wrongly the sensors data, due to a lack of understanding or information about the monitored system. To theoretically

[1] Network Intrusion Detection System.

avoid false-positives, a filter would need to know continuously the state and behavioral rules of the end-systems that generated these data. Such a level of knowledge can not be reached in practice, as shown in § 3.1.1. Hence it is theoretically impossible to totally eradicate false-positives.

As time is passing, IDS have to monitor a constantly increasing flow of information, thus generating an increasing volume of alerts. Network based sensors will be required to monitor high volume of network traffic, and host based sensors may have to monitor larger systems. The progressive integration inside distributed IDS of many different sensors will multiply the volume of alerts to treat at a central monitoring point.

Although the quantity of attacks may increase, due to the spreading of attack tools and the decreasing of the competence level required to run attacks, it can be expected that normal activities will increase faster. Beside, normal activities generates false-positives at a constant rate. As a consequence, the ratio of relevant alerts per false-positives may decrease with time. In other words, IDS will probably have to deal with a statistical rise of the percentage of false-positives inside the alert-flow.

3.2.2 Hiding attacks

There are techniques to hide an attack inside the alert-flow. We already introduced one of them: evasion. Some others are based on false-positives flood, or relevant alert frequency and sources. New techniques may appear with time, and those presented here do not make an exhaustive list. Our purpose here is to show that attacks may be hidden from IDSs.

3.2.2.1 Evasion

As seen previously in §3.1.2, evasion allows to transmit data over a network without an IDS analyzing it.

3.2.2.2 Alert flood

An other way of hiding an attack or just disabling momentarily an IDS consists in flooding the IDS monitoring central and its end-users with a huge amount of false-positives. This is called alert flooding. It can result in a Denial of Service of the IDS if it is not able to efficiently contain the flood of false-positive. It can indeed get hard to detect relevant alerts among huge amounts of false-positives. An attack can be hidden in this way from the IDS. Yet the alert flood in itself is an information that something is going on, and should motivate further inquiry.

Making an IDS generate alerts is technically easy: most of the time, the attacker does not need to perform an attack. Just sending for example forged TCP packets with typical attack signatures in their payload works against most current IDS (those that do not perform TCP stream reassembly). There are automated IDS testing tools based on this principle, that can be used to generate alert floods (IDSwakeup).

3.2.2.3 Slow rate attacks

An attack is often done in many successive steps. It might be possible to hide an attack by performing each of these steps after some delay. In this way, even if the IDS raise an alert for some of these steps, they might seem unrelated to the end-user, because they are too delayed in time.

3.2.2.4 Distributed attacks

A last technique consists in making an attack hard to detect through using different sources for each of its successive steps: this is a distributed attack. The challenge for the end-user here is to make the correlation between apparently unrelated alerts.

3.2.3 Pressure on the end-user

The problems related with alert-flow listed in previous sections (§3.2.1 and §3.2.2) are all pointing out one main issue with IDSs: being able to efficiently manage the alert-flow in order to extract relevant information from it. This ultimately puts a high pressure on the IDS monitoring end-user. He/she needs to be competent enough to interpret the alerts displayed by the IDS, perform alert correlation, and take the appropriate measures. The required skills (experience, knowledge and resistance to boredom) are hard to find and develop, which may result in a shortage of educated monitoring personal. Such personal will consequently get expensive, and be harder to afford for small companies.

3.3 Sensor's technical requirements

IDSs have to be able to monitor a constantly increasing amount of data, which pushes IDSs to become more efficient.

3.3.1 Efficient TCP/IP stack

Technological progress is constantly increasing the rate of network traffic. Network based sensors have therefore to adapt and be able to handle high rate traffic with a minimum packet loss. On very high traffic sites, such as gigabit Internet backbones, it is possible use load balancing to distribute the traffic between multiple sensors. But distributing data streams among IDSs introduces the problem of correlating the detection between the filters of these different IDSs.

Besides, with regards to the problems related to TCP/IP stack implementation exposed previously in §3.1.1, the TCP/IP stack used in the sensor to reassemble traffic should meet quality requirements not only in terms of speed, but also in term of compliance with the protocols official descriptions. This mainly means efficient IP fragment and TCP stream reassembly.

3.3.2 Efficient filters

To follow the increase of traffic rate, filters have to run efficiently in order not to become a bottle neck for traffic analysis. Filters are thus required to use little memory and CPU time, to spare system resources.

There are usually two alternatives for implementing filters: either they are compiled as assembly code, or as a byte-code and interpreted by a filter engine inside the sensor. Compiled filters will be faster than interpreted ones, since they precisely don't require interpretation. Compiled filters can also be optimized on an assembly level to increase efficiency. Interpreting filters on the other hand, while offering more flexibility for writing filters through the use of an intermediary and specifically designed language, puts high efficiency requirements on the interpretation engine.

3.4 The cryptographic challenge

The increasing use of encrypted traffic in networks is slowly becoming a deadly obstacle for network based IDSs. More and more network traffic is now going through encrypted tunnels which put the traffic out of reach of network based sensors. Network based IDSs will still be able to access and control the protocol header part of data packets of transport protocols used to carry encrypted data, such as IP or TCP, but will not have access to their encrypted data payload.

Yet, there is a way of bypassing encryption: by using an encryption proxy at one end of the encrypted tunnel, typically at the server side. Hence, all traffic is encrypted between the client and the proxy, but decrypted between the proxy and the server. The IDS's sensor can then be placed to listen on this decrypted traffic. It is for example the only currently available solution for monitoring HTTPS/SSL traffic. Yet, placing encryption proxies in front of every server using encryption is a hard constraint in a network architecture, and does not appear as an optimal solution.

3.5 Learning from the antivirus world

An interesting parallel can be drawn between the evolution of viruses and antiviruses, and the world of attacks and IDSs. This can give us an insight on what the future of IDS will probably look like.

3.5.1 Similarities between antivirus and IDS

Virus detection presents many similarities with attack detection. A virus can often be identified through a signature, like most of the attacks. Antiviruses have therefore been using signature detection from the beginning, as IDSs did. A virus acts unconventionally as a program. For example, it replicates its own code and modifies other executables, or modifies master boot records, which usual programs don't tend to do. A virus's activity can thus be detected with anomaly detection techniques. Antiviruses indeed use anomaly detection, although they call it 'heuristic analysis'. Besides, most

of the techniques used in viruses and antiviruses design have an equivalent in the IDS world.

Viruses and antiviruses have been studied and produced for more than 10 years, and are now technically mature, while IDSs are still in their youth. Antivirus systems have been available commercially many years earlier than IDSs, and it can be assumed that IDSs will follow the same evolution at least on a few aspects.

3.5.2 The race for polymorphism

The first antivirus programs were signature based. In the early days of viruses, all the copies of a virus had indeed the same body, and it was therefore possible to find all the copies of a virus spread inside an infected system just by searching systematically for an occurrence of a specific part of the virus's body.

To escape antivirus detection, many technics were developed by virus writers to reduce and finally remove any similarity between one virus and its copies. The idea was at first to encrypt the body of the virus and let only the decrypting procedure in clear. Then were developed 'mutation engines' that were able to generate a functionally identical decryption procedure, but encoded differently in terms of assembly instructions. Mutation engines are exploiting the fact that a same operation can be written in many different ways at the assembly level. Viruses using mutation engines are called polymorphic viruses, and are impossible to detect with raw signature detection since there is no constant similarity between two copies of the same virus. In reaction to this evolution, antiviruses started using heuristic analysis, which is the antivirus equivalent of IDSs' anomaly detection. Such antiviruses monitor the activity of programs on systems in an attempt to detect behaviors that are typical of viruses

The interesting point is that attacks are following the example of viruses with a few years delay, and are currently just starting to explore the possibilities of polymorphism. There are nowadays attempts to write polymorphic attacks[2], or tools for generating polymorphic buffer-overflows[3]. These technics are still very young, but can be expected to grow quickly. IDS that are exclusively using signatures will then become inefficient against a wide range of attacks.

3.5.3 Evasion tricks

While seeking polymorphism, viruses were adapting technics to evade antivirus detection using system breaches and tricks to become stealthier. Some viruses are even trying to detect an antivirus's presence and consequently hide themselves or attack the antivirus. Most of the changes between virus generations are focused on developing and exploiting new tricks to evade detection. As a result, viruses and antiviruses are engaged in a constant race between evasion and detection.

In much the same way, attacks are now trying to exploit system or IDS breaches in order to evade detection (see section §3.1.2). Considering the importance of such techniques in the virus world, we can expect them to become equally important in the

[2] Exploiting IIS's unicode bug for example.

[3] *ADMmutate*, written by K2, released in march 2001.

IDS world. In particular, attackers have not been focusing so much yet on trying to identify IDSs and exploit their specificities to evade them. Hence the importance of being stealthy for an IDS (see §2.8.4). Moreover, a race between evasion and detection implies a constant need for research and reaction from the IDS development side, which should be taken into account when planning long term resources for IDS development. Finding ways of handling new attacks as automatically as possible is therefore an issue.

3.5.4 Superiority of the heuristic approach

The antivirus equivalent of anomaly detection is called heuristic analysis. It covers a range of system monitoring techniques aiming at detecting signs of a virus's activity. In this approach, a model is built describing viruses typical activities, and the system is monitored in order to detect programs behaving according to this model. From a certain point of view, anomaly detection works in the opposite way, by detecting activities that are not belonging to a model of 'normal usage'. Yet, the implementations of these two techniques are very similar, and it can be assumed that they have an equivalent efficiency.

The heuristic analysis has been proven technically efficient for detecting viruses, especially viruses that have not been classified yet and for which signature based antiviruses still do not possess any signature. Yet, heuristic analysis has two drawbacks that justify their lack of success as commercial products:

1. They do not provide clear alert information. Where a signature based antivirus can provide an alert specifying clearly which virus has been detected and how to remove it, heuristic engines would send alerts describing an anomaly in program behavior, which can be hard to interpret for non educated users. This does not fit with the aim of most antivirus products to reach a large market, and thus users without a deep computer experience.

2. They are commercially less profitable than signature based products, since the user don't have to subscribe to periodical signature database updates.

Antivirus softwares are now seldom heuristic based only. The tendency is rather to rely on signature detection, while simultaneously running an heuristic engine in order to automatically detect new viruses. Suspicious programs are then automatically sent to the antivirus company for a further analysis, which will eventually lead to a signature release.

If we perform a raw transposition of the lessons of heuristic analysis to anomaly detection, we can project the following results:

- Anomaly detection works, but requires an educated monitoring. Signature based detection should be preferred when the monitoring personnel is lacking computer experience.

- Anomaly detection could be used in parallel with signature detection, thus allowing to develop an automated process for detecting, analyzing and generating signatures of new attacks. This would be a first answer to the polymorphism and evasion problems.

Chapter 4

Designing efficient filters

A first approach to solving some of the problems presented in chapter 3 is to improve the quality of filters. The author has been working 5 months on writing anomaly filters for a network based IDS, and is exposing in this chapter the results and techniques he has learnt through this work. The chapter contains a discussion on efficient filter design and two concrete examples of filters. We will only consider filters for network based IDSs.

4.1 Development strategy

4.1.1 Advantages of efficient filters

We could define an efficient filter as a filter which 'understands' the data it analyses and generates relevant alerts and limited amount of false positives.

Efficient filters will not solve the problems of encrypted and high rate traffic, but they might bring improvements to problems related to analyzing the data and alert flows. The problems related to intrusion detection (insertion & evasion, false positives, slow and distributed attacks) and to alert flow processing (false positives, alert correlation) can partially be addressed at the filter's level.

From a general perspective, the more accurately a filter 'understands' the data, the more efficient it can be. Pushing the idea to its extremity would lead into designing filters that are small expert systems. This may not either be desirable considering the filters efficiency requirements expressed in §3.3.2. A right balance has to be reached.

4.1.2 How to make an efficient filter

A first issue is to clearly define which data the filter should analyse and which not, so to set boundaries to the filter's area of responsibility. Designing filters on a protocol basis has proved to be efficient, since it allows to straightforwardly identify which filter to update in answer to new attacks and enables the integration of a protocol anomaly detection module in the filter. Of course, this should not be an absolute rule: some

protocols (HTTP, TCP) may require a wide set of filters for them alone, and some filters may be required that does not fit into a protocol based approach (detecting vulnerability scanners for example). In this chapter we focus on protocol filters.

An other issue is to integrate both signature and anomaly detection techniques in the filter. As seen previously, both techniques are complementary and gain at being used together. For each protocol, a part of the corresponding filter should analyse the protocol packets and check their compliance with the protocol. Actually, practice shows that this protocol analysis capability enables a more accurate signature detection, since the signature can be defined precisely and checked for at the exact places.

Finally, techniques should be used in the internal filter design to reduce the amount of false positives and try to generate high level and highly descriptive alerts, as exposed in §2.4.2.3.

4.1.3 Developing protocol filters

The development of a protocol filter is based on a few successive steps. The first step is to gather and read all official documentation about the protocol, which includes RFCs and Internet drafts. The second step is to identify all the aspects of the protocol that should be checked in a protocol anomaly detection engine (headers, fields, etc.). The third step is to gather the list of all known attacks using this protocols. Web sites such as securityfocus or attrition.org[1] can be used, as well as books[2].

Then comes a design stage, in which choices have to be made between what the filter will actually check, and which attacks or anomalies we will choose to ignore. Such decisions are inspired by a compromise between the filter's resources (speed, memory usage, etc.) and its efficiency at detecting intrusions. Finally comes the programming and testing.

Some programming issues have been presented in §2.4.2.3, with the possibility of integrating alert flow filtering components inside the filter itself to perform a first level of correlation. An other issue is to adopt a design clear and flexible enough to enable smooth updates of the filter.

4.1.4 Comment about the examples

The two filters that are described in §4.2 and §4.3 have been implemented at Defcom Security in Stockholm, as part of Defcom's IDS, OnGuard. The filters were written in n-code, a language used for programming the sniffer engine developed by Network Flight Recorder (NFR). Their sources will not be released here, for reason of industrial secret, nor will their implementations be discussed in detail. They are used as short examples to illustrate the issues related to efficient filter programming.

[1] www.securityfocus.com, www.attrition.org
[2] 'Hacking exposed'. See bibliography.

4.2 Detailed example: a telnet filter

We illustrate here a way of writing a filter for the telnet protocol. This filter checks for most of the known telnet related attacks and does protocol anomaly detection.

4.2.1 Protocol and vulnerability analysis

Protocol anomaly detection

The telnet protocol is being described through more than 38 RFCs, but the most important for us are RFCs 854 to 861 (describing telnet and its main options), 1073 (window size option), 1408 (environment option), 1409 (authentication option) and 1572 (new environment option).

Reading these RFCs reveals that a protocol anomaly engine for telnet should be able to separate the telnet stream into the user data stream and the telnet option negotiation stream. Telnet options are special data inserted into the telnet traffic, and used by the client and server for mutual configuration. They are always preceded by an escape character, and have a standard format (RFC 855). The anomaly engine should be able to disassemble the option headers and analyse them to check that they are compliant with the protocol (standard option code used, valid syntax, etc.).

An other anomaly not specified in RFCs but inspired by the practical use of telnet is the use of extremely large or not printable usernames or passwords. The anomaly engine should hence be able to identify usernames and passwords, and check if they have a reasonable length and do not contain binary data.

Vulnerability analysis

A survey among vulnerability archives reveals 17 attacks using telnet itself. Below is a list of these attacks, sorted in four categories:

Buffer overflow on username or password:

- Rideway PN dos (buffer overflow on beginning of connection)
- Foundry network (long password + enter = reboot switch)
- GAMSoft telnetd (username of 4550 chars)
- Shadow Op Dragon telnetd (DoS if username > 16500 chars)
- GoodTech Telnet Server (DoS if username > 23870 chars)
- ByteFusion Telnet (DoS if username > 3090 chars)

Other buffer overflows:

- MS hilgraeve hyperterminal (telnet address of more than 153 chars, triggered through html)

- Windows 9x Telnet Client (buffer overflow on 'connect failed msg', triggered through html)

- Win2000 telnetd (send a stream of 0)

Magic keywords

- netopia 650 (displays system logs if ctrl-F or ctrl-E typed)

- freebsd telnetd (telnetd searches for file with TERMCAP before login, thus creating heavy overload)

- IRIX telnetd (execute commands embeded in IAB-SB-TELOPT_ENVIRON)

- cisco IOS telnetd (when ENVIRON used, IOS reboots)

Bad programming

- cisco catalyst memory leak (telnet server does not free mem after usage: dos)

- cisco online help (cisco's access lists accessible to non authorized users through badly implemented cmds)

- MS win2000 telnet.exe (always try to connect with server with NTLM: may be sniffed & cracked)

- win2000 telnet session timeout dos (server does not timeout and wait for user identification)

A choice has to be made as to which of these attacks the telnet filter should be able to detect.

The two buffer overflows related to microsoft's hyperterminal and telnet client are triggered locally, and therefore can not be checked by a telnet filter. They should eventually be filtered by an HTTP or HTML filter, in case the html page carrying the attack is transfered through a monitored segment.

Attacks related to bad programming usually make a correct use of the protocol, and thus should not either be checked inside the filter. The responsibility of the vulnerability belongs to the software used, not to the protocol. Besides, trying to efficiently detect these attacks would require the filter to know the software and hardware versions of every telnet component used, and this knowledge would be very hard to maintain in practice.

Finally, the detection of some attacks may require an unacceptable overhead in filter resources or complexity. Some attacks may also be so old that it can be reasonably assumed that the corresponding vulnerabilities have been fixed and the attacks can not be used any more.

Additional feature

Once a user has been successfully authenticated in a telnet session, it has theoretically the right to do whatever he/she desires on the obtained shell. Yet, the user session could be monitored for some typically suspicious activities (such as the user trying to access /etc/passwd on a unix system). We could thus implement a simple command checking engine, triggering alerts when the user types one of a few suspicious commands on the telnet shell.

And we should not forget the most straightforward of all attacks: brute forcing, when someone tries systematically to login using various username/password combinations.

4.2.2 Design

Most of the attacks found could be handled by searching for some signature in the telnet packets. This kind of approach has known drawbacks: it is indeed easier to implement, but hard to update, and unable to 'understand' the protocol, thus inefficient in complex detection situations and susceptible to generate false-positives.

A better approach is to integrate both an anomaly and a signature engine to the filter. The filter should be able to disassemble the telnet stream and divide it into two streams: a telnet option stream and a user-server data stream. Anomaly detection would be performed on the telnet option stream to check for compliance with the telnet protocol, and on the user-server stream to monitor the login stage. The user-server data stream would pass through the command checking engine.

The filter should furthermore be able to differentiate the login stage of a telnet session from the 'user logged in' stage, and maintain internally a list of failed login attempts with their originating ip addresses, in order to enable brute force detection.

Some of the attacks listed in §4.2.1 can not be detected otherwise than with raw signature detection (attacks using ENVIRON option, for example). Yet, this signature matching can be performed much more accurately thanks to the anomaly engine in charge of analyzing the option stream. More accurate signature matching also reduces false positives. We could for example check for ENVIRON related attacks only in the ENVIRON option header when it occurs, and not systematically through all telnet traffic, as most of the filters currently do.

Figure 4.1 shows the structure adopted for the telnet filter. A *Data Parser* separates telnet options from user data. Telnet options are checked for their compliance with RFCs by an *Option Validity Checker* and a *Sub Negotiation Validity Checker*. Depending on whether the user is authenticated or not, data is passed through a *Login Checker* (checking for long or non printable usernames and passwords) or a *Good Usage Monitor* monitoring the user activity. A *Brute Force Monitor* runs in parallel, in cooperation with the *Login Checker*.

4.2.3 Implementation

There are a few programming issues rising while implementing the structure described on figure 4.1.

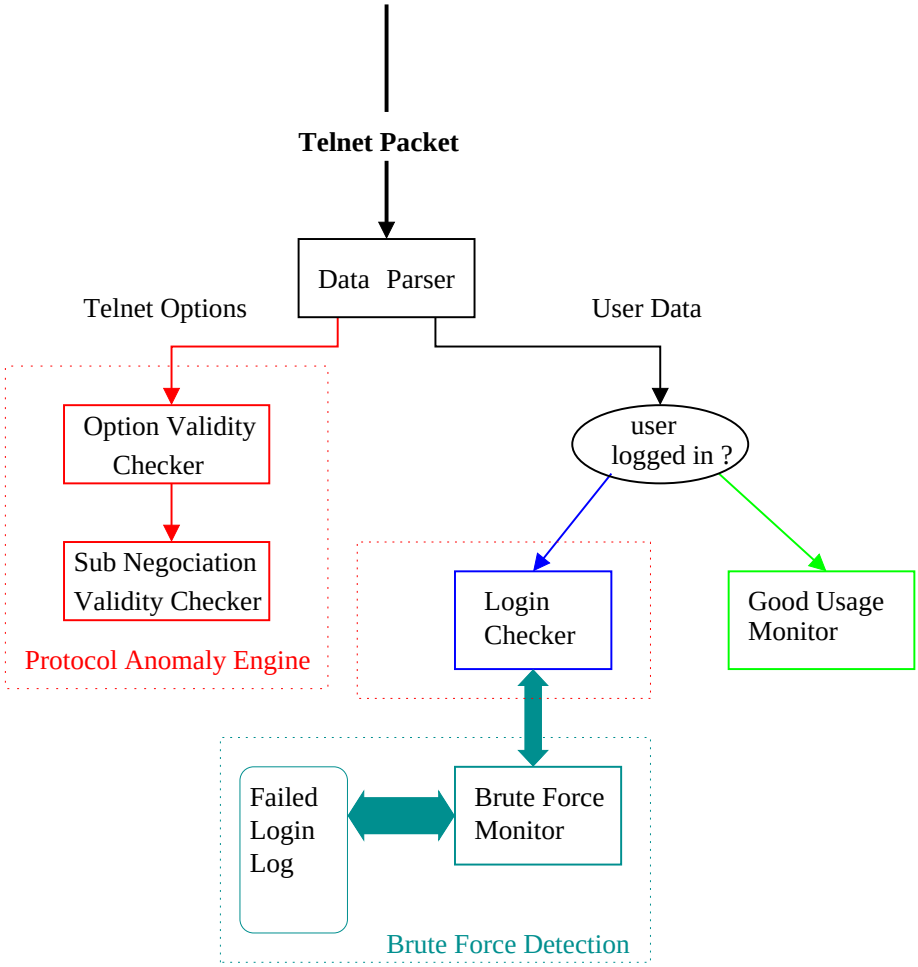

Figure 4.1: Structure of the telnet filter

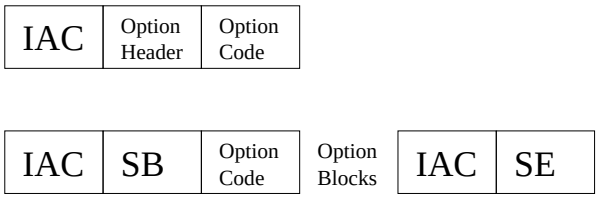

Figure 4.2: Telnet option format

Separating telnet options from user data

The telnet protocol specifies two possible format for telnet options (RFC 855), as shown on figure 4.2. To separate telnet options from the data flow, one just need to intercept the escape character IAC (ascii 255) and its following byte (WILL, DO, WONT, DONT or SB) in order to know how to handle the option block.

Differentiating username, password and shell prompts

The telnet protocol does not specify any standard username and password prompt. The filter could simply check if the server's answer ends with the string 'username:' or 'password:', but this will not work with all servers. An alternative is to assume that the password and username prompts end with ':' while the shell prompt ends with '>' or '#'. But again it won't work for all servers.

The solution chosen for the telnet filter was to assume that the first server answer is the username prompt, and the second the password prompt. If the third answer is different, it means the user has logged in, and if it ends with the same string than the former username prompt, it means the login failed.

Which anomalies to detect

Anomaly detection should be performed wherever it is required, at any place in the structure of the filter. As an example, the telnet filter implemented at Defcom reacts on the following anomalies:

- End of subnegociation block not found (the IAC SE sequence has not been found).
- Unrecognized telnet option code or unrecognized option header. Recognized option codes are: NOP, DM, BRK, IP, AO, AYT, EC, EL, GA, and recognized option headers: DO, WILL, DONT, WONT, SE.
- Too long username or password.
- Binary code in username or password.

Brute forcing detection

Which criteria should be used to trigger a brute force alert ? A way is to rise an alert when either a given user has failed to login more than a given amount of times N1 during a certain period T, either the total amount of failed logins during T exceeds a value N2. N1 and N2 should be chosen depending on the activity of the telnet server monitored. N1 could also have a higher value for normal users than for privileged users.

4.3 Second example: TCP filter

The second example is a protocol filter for TCP. Considering the complexity of the filter, it will not be detailed as much as the telnet filter.

4.3.1 Protocol and vulnerability analysis

TCP is a complex protocol, and entering in the details of implementing a TCP filter is beyond the scope of this report. We can just give a short list of the main issues addressed by this filter:

Anomaly detection

The filter should parse the TCP headers of each TCP packet, and check that reserved flags are nulls, URG flag and urgent pointers are used simultaneously and that the TCP options are valid. The filter should maintain a log of all TCP exchange, and monitor in particular invalid or unfinished session openings or closings that are typical of portscans. It should monitor window sizes and TCP buffer usage.

Attack detection

There are many different kind of attacks based on the TCP layer. The filter should contain a signature engine for detecting DoS attacks such as land, latierra and winuke. It should be able to detect ACK storms, that are a sign for TCP hijacking. And it should analyse portscans accurately, through the detailed log of each TCP exchange. An extensive documentation on portscans can be found in the man page of the 'nmap' tool.[3]

TCP hijacking is in itself a hard issue, since it does not have systematic symptoms. Normally, TCP hijacking can be detected through the ACK storms it creates, but the hijacked system can be at first DoS-ed to avoid it from starting the storm.

4.3.2 Implementation

Figure 4.3 shows the simplified structure of the TCP filter. The filter is mainly structured around three components: a *Header Sanity Checker* monitoring TCP headers for

[3] *'nmap'* (www.insecure.org/nmap) is the most extensive unix/linux portscanner, written by Fyodor.

anomalies, the *Session Monitor* keeping data on every ongoing TCP exchange and the *Scan Detector* analyzing the session logs to detect portscans. In addition to these three modules, a DoS Checker is used to detect known DoS attacks.

The *Session Monitor* includes a *Session Anomaly Checker* performing anomaly detection on a session level. The *Scan Detector* includes a *Dead Session Monitor* to check for timed-out TCP connections.

4.3.3 Results

The TCP filter proved to be extremely accurate in its analysis of anomalies and portscans. It can for example make the difference between a normal scan and an OS fingerprinting scan. The filter does not solve on the other hand the problem of slow or distributed portscans. Yet, if configured to be extremely sensitive to portscans, the filter can be used as a first detection layer sending alerts for even very small scans concerning only a few ports for a given period of time. Later on, a data mining engine can analyse these alerts and detect the slow or distributed scans.

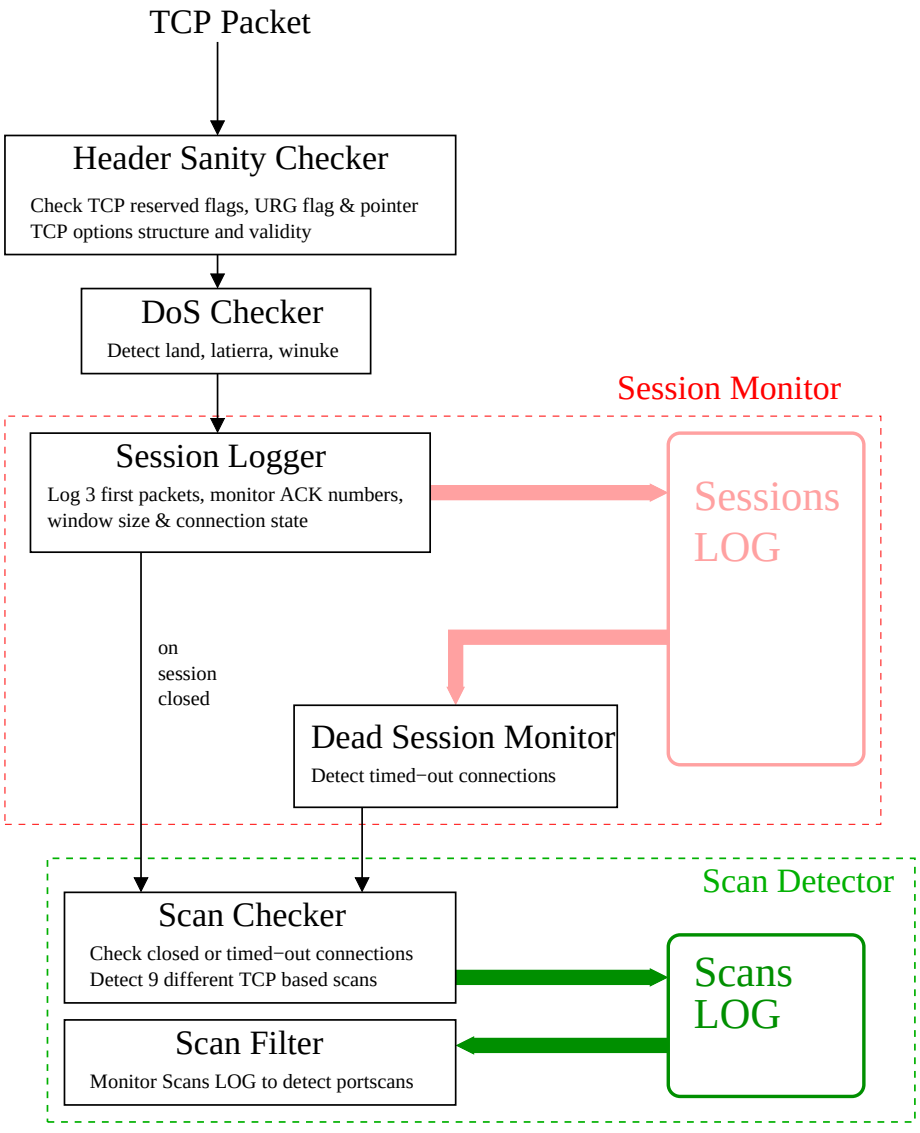

Figure 4.3: TCP filter architecture

Chapter 5

Suggested architecture for a long term competitive IDS

Chapter 4 shows that designing more efficient filters may improve IDSs, but does not bring solutions to most of the issues presented in chapter 2 and 3. We may therefore try an other approach and reflect on how the IDS classical architecture could be changed in order to meet these issues. This chapter describes such an alternative IDS architecture.

5.1 Requirements for a long term competitive IDS

We can now establish a list of requirements for an IDS aiming at being competitive on a long term basis. At first, the IDS should efficiently detect attacks, which means that:

1. The IDS should always send an alert when an attack is performed. This should not depend on wether the attack is well-known or not.
2. The IDS should rather not send an alert when no attack is performed. As seen previously, false-positives can not be avoided, but their amount can be reduced.

With regard to the issues presented in chapters 2 and 3, this IDS should also as much as possible:

1. Resist to insertion and evasion.
2. Process efficiently the alert-flow. This include efficient filtering of false-positives, resistance to alert floods, and detection of slow rate or distributed attacks.
3. Be able to handle high traffic rates.
4. Monitor activities carried through encrypted tunnels.
5. Resist to polymorphism.
6. Be able to carry up with the booming multiplication of new attacks.

7. Be friendly to the end-user.

Finally, it should finally follow the 3 security requirements exposed in §2.8.1:

1. IDS components as well as the data they exchange should not be accessible to unauthorized users.
2. If a component of an IDS gets compromised, it should not compromise any other part of the IDS.
3. If the IDS is compromised, it should not compromise the security of the monitored system.

Other requirements could be seen as useful, as for example frameworks for administration, update and configuration, or shunning capabilities. Yet, these functions can be added as peripheric components to the IDS.

5.2 Suggested architecture: MIDS

5.2.1 Description

Designing an IDS meeting the requirements listed in § 5.1 is a challenging problem. The architecture suggested here does not pretend to bring solutions to all these requirements, and may present some serious implementation difficulties. Yet, it will be demonstrated that it theoretically solves some major issues, and is believed to be close to an optimal solution.

The idea is to build an IDS based on both host and network based sensors. Network based sensors should be of different kind, each one using a specific TCP/IP stack implementation, so that the sensors all together cover a representative panel of stack implementations. Host based sensors, beside their usual monitoring functions, should focus on controlling the activity generated by network server applications. Both host and network based sensors should use simultaneously signature and anomaly based filters. All alerts generated by the sensors should be merged in a common alert-flow, and if necessary converted to a common alert format. A specifically designed dynamic correlation engine would process the alert-flow in order to increase its relevancy. Finally, the sensors should be interconnected through an isolated and secured communication network.

Other components, such as a GUI to parse alerts, various administration frameworks, passive alert correlation engines, shunning functions or report generators could be added later on to this core structure.

5.2.2 Implementation

A way to implement such a structure is to build a 'Macro IDS' (MIDS), based on currently available IDSs, as shown in figure 5.1. The problem is then reduced to agreeing on a common alert standard, writing alert-conversion plugins to various IDSs, and building the correlation engine. How to build the MIDS and its technical advantages will be further developed in § 5.4.

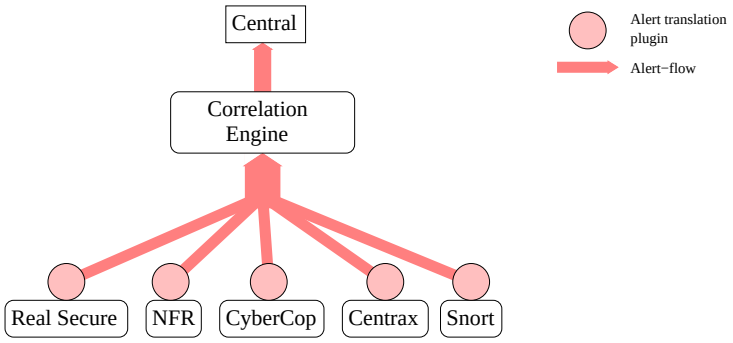

Figure 5.1: MIDS architecture

5.3 A different vision: alert-flow control and intelligent processing

The structure shortly described in § 5.2 is based on a different conceptional approach to IDSs.

We suggest to consider IDSs as information flow conversion and management systems, as shown on figure 5.2. In this regard, it becomes important to have access to the most representative set of data describing a monitored system's state and activity, hence the multiplication of sensors and sensor types. Filters appear as components converting the information flow, changing sensor data-flows into alert-flows. The correlation engine then stands as a component to dynamically reshape and improve the alert-flow.

Of all the requirements listed in §5.1 related to future IDS challenges, we will show that they can either be solved by technically improving the sensor or improving the information-flow. This informational approach precisely emphasizes the role of the filters and the correlation engine as data-flow and alert-flow processing components.

5.4 MIDS detailed architecture

This section describes in more details the structure and implementation of MIDS, and shows how each aspect of MIDS's structure is aiming at enhancing some of the requirements for efficient IDSs listed in §5.1.

5.4.1 Based on multiple IDSs

5.4.1.1 Multiple sensor type

MIDS should use both host and network based sensors, in order to gather complementary information from the monitored system. Complex attacks often consist in a part performed remotely over a network and a part performed locally on a compromised server.

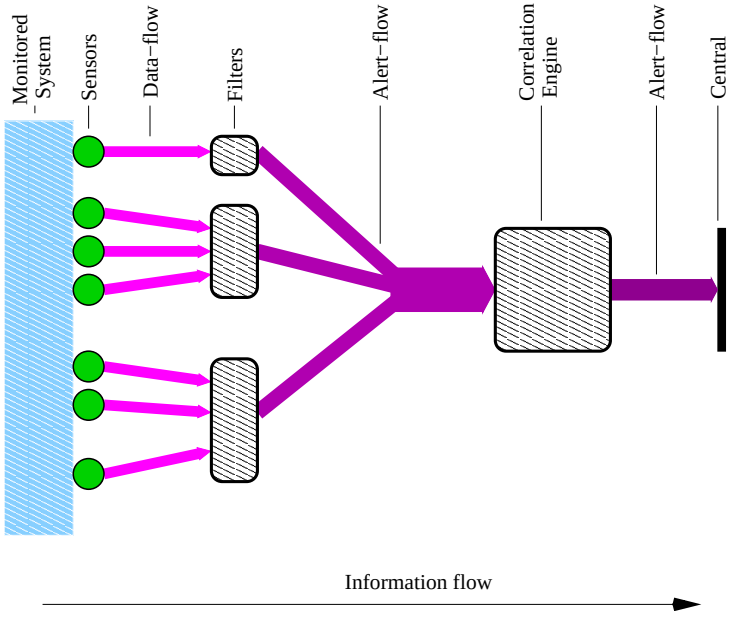

Figure 5.2: IDS as an information flow conversion & management system

Attacks performed over a network can usually be detected with network-based sensors (ex: remote buffer overflow).

Yet, network-based sensors should not be used alone: host-based sensors are required to detect local attacks (ex: local privilege escalation). Host-based sensors are also needed as a second line protection against insertion/evasion technics. Attacks using such technics may evade a network-based IDS, but not an host-based IDS when performed locally.

Furthermore, host-based sensors may bring a solution to the cryptographic challenge. The encrypted traffic between the client and the server can not be monitored for attacks (ex: unicode bug inside https), but some signs of an attack could be detected on the server side with a host-based sensor.

5.4.1.2 Multiple TCP/IP stack implementations

Insertion and evasion are partly the consequence of the differences between TCP/IP stack implementation. MIDS could circumvent this problem by using simultaneously multiple network-based sensors, each with a different TCP/IP stack implementation, in such a way that all the different stack implementations in the monitored system have one representant in the MIDS. Each network-based sensor in MIDS could be configured to specifically monitor only the hosts having the same stack implementation.

In practice, such a requirement may be hard to follow: it is hardly feasible to keep track of all (undocumented) changes in stack implementations, and NIDSs tend to use their own proprietary stack implementation.

5.4.1.3 Mixing filtering techniques

Mixing anomaly and signature detection has been proved to be efficient (see chapter 4).

Anomaly detection enables accurate signature detection, detection of unknown or polymorphic attacks (heuristic analysis has indeed shown its efficiency against polymorphism in antiviruses). Besides, signature filters can give detailed information on the nature of an attack, thus being more end-user friendly. Signature and anomaly detection are therefore completing each other, and should be used simultaneously.

5.4.1.4 Using readily available IDS

Developing an IDS with host-based and network-based sensors, using multiple stack implementation and multiple filtering technics may be hard to realize in practice. A possible short-term compromise would be to use a panel of well-chosen currently available IDSs and merge their alert-flows. This would be cheaper and faster to implement, while providing at least the level of efficiency of an IDS alone.

5.4.2 Merging alert-flows

Using many IDSs in a common monitoring architecture implies to have a way of merging their alert-flows

There are two ways to achieve this state: either by developing a common alert standard followed by each of the IDSs used, or by adding to each IDS used a component translating the alerts from their initial proprietary format to the format used by MIDS. There are currently attempts to define common alert standards, but none has emerged as a widespread standard. Besides, the difficulty for the various IDSs manufacturers of reaching a common agreement makes this first option quite utopian on the short-term. Designing specific plugins for each used IDS therefore appears as a simpler alternative on a short-term sight.

Defining a general alert standard able to describe alerts coming from any kind of IDS is an important issue here, and the candidate that is the most likely to be chosen is IDMEF/IDXP, which RFC should be released by the end of 2001.

Yet, merging the alert-flows implies more than just converting the alerts to a same format. Alerts contents need to be analyzed and translated into a common alert description syntax. This is a complex process, requiring the use of a correlation engine.

5.4.3 Correlation engine for intelligent alert-flow processing

The correlation engine dynamically processes the alert-flows produced by different sources to properly merge them and increase the relevancy of the common alert-flow.

The dynamic aspect of the correlation engine sets a restriction over the delay during which the engine is allowed to keep an alert without forwarding it. This consequently limits the quality of the analysis. A correlation engine could at first perform a few simple operation on the alert-flow, in order to more accurately merge alert-flows from

different sources and filter and re-shape the common alert-flow. Below is a non exhaustive list of such features.

Alert Standardization Alerts coming from different DSs but describing a same event should be translated into a same common alert. This completes the process of merging alert formats, as explained in the previous section.

Redundancy Avoidance When merging alert-flows from different sensors or different filters in a same sensors, it may occur that multiple equivalent alerts are emitted. This overload could be avoided by merging equivalent alerts in one unique alert.

Common Alerts Summary Some alerts may be sent at high rate, thus flooding the end-user. An alert summary engine would periodically cache these alerts and replace them by a macro-alert summarizing their contents.

Knowledge Addition It may in some cases be possible to increase alert relevancy by using additional knowledge on the monitored system.

False-positive Filter Well-known false-positives can be removed by a component using a specific scripting language to define alert filtering rules.

Filtering Level A filtering policy with multiple levels could be defined. Alerts could then be forwarded or summarized depending on their severity or their type of activity, thus implementing a level of filtering.

The correlation engine could then eventually perform some more complex operations, such as:

Scenario Recognition An attack may generate multiple alerts, especially if anomaly detection filters are used. Some attacks can therefore be recognized through the specific alert signature they generate. A scenario recognition component could help identifying ongoing attacks by dynamically searching for attack signatures inside the alert-flow. This would also make alert messages more understandable to the end-user, by more clearly identifying suspicious activities. Vulnerability scanners or complex scans could be identified in this way. Yet, this technic is challenging to implement in a dynamic way, and requires the difficult creation and maintenance of an attack scenario database. Yet, research projects are progressing in this direction, such as the French project 'Mirador'.

DDoS Detection & slow attack A specifically designed component could monitor Denial of Services in order to detect distributed ones.

Anomaly Detection Statistical anomaly detection technics could be applied to the alert flow itself, in order to detect unusual alert patterns in the alert-flow and alert about them. Yet experience shows that an alert-flow is often very irregular, thus making this technic hard to apply.

Flood Regulator Using an anomaly detection component, it would be possible to detect sudden alert-floods and apply stricter filtering rules when detecting one to stop or summarize less important alerts.

Long Alert-flow Caching A component could monitor the alert-flow on a longer time-scale by caching the alert-flow a longer time. It would enable more effective scenario recognition, or recognition of low rate or distributed attacks. This however could be done with passive correlation on the alert database level.

The implementation itself of the correlation engine is challenging: some components could be designed as compiled stand-alone alert-flow proxy components, others may require high level description or scripting languages. When using scenario recognition, data-mining or knowledge addition, the correlation engine gets closer to an expert system and may need artificial intelligence technics.

5.4.4 Isolated inter-IDS network

Isolating the alert-flow and management side of the multiple IDSs inside a closed inter-IDS network would enable the enhancement of a higher security level, as described in § 2.8.1. IDSs' sensors should be the only parts of MIDS directly connected to the monitored systems, either as promiscuous network interfaces or as daemon processes on hosts. Each sensor should be designed in such a way that an attacker compromising one host-based sensor cannot go further in the MIDS, and that an attacker having compromised the MIDS cannot compromise the monitored system.

But these structure is hard to implement while using multiple IDSs whose security models have not been designed specifically for MIDS. We therefore have to rely on each used IDS's own security.

However, this secured architecture can be built starting at the alert-flow conversion plugin in each IDS. Each conversion plugin should be connected to an isolated network whose entry points are the plugins, and which securely forwards the alert-flow and management traffic, and hosts MIDS components such as correlation engines, administrational framework and GUIs.

Cryptography could be used to securely identify, authenticate and communicate between components in this inter-IDS network. Network based sensors should be located on dedicated computers whose network interface has been altered on a hardware level to forbid data emission. Host based sensors should be self monitoring, in order to detect attempts to disable them.

5.5 Unsolved problems

The suggested MIDS architecture however does not bring solutions to all the problems listed previously.

Network-based IDSs will still need to resist high traffic rates.

The efficient handling of new attacks could actually be improved through the correlation engine, by using it for detecting new attacks and trigger a semi-automated process of attack analysis. The same technique has been successfully applied to antiviruses.

MIDS would not be stealthy, since it would use host-based sensors. Yet, the interest of IDS stealthiness was to avoid an attacker to use specific evasion technics, which should anyway be inefficient against MIDS.

The largest remaining problem is actually to implement MIDS. The process of integrating proprietary software in a common architecture will quite likely face difficulties of many kind. Besides, the correlation engine is a challenging component and an open problem.

Chapter 6

Conclusion

IDSs are facing hard challenges: evasion techniques, cryptography, false positives and high rate traffic among others.

Solving these difficulties can partly be done by designing efficient filters using both signature and anomaly detection, and integrating some layers of alert-flow processing. An alternative is to focus on improving the structure of the IDS. We suggested the MIDS architecture, which is a general alert-flow processing system using inputs from multiple IDSs, both network and host based, and focusing on intelligently merging and correlating their alert-flows. MIDS is about to become reality. Many research groups are working on developing similar architectures, such as the Mirador project in France or OnGuard at Defcom.

Besides, we reached a new vision of IDSs as information flow processing systems. In this perspective too, the correlation engine is becoming the key component in the future of IDS design. Building an efficient correlation engine can be expected to be a challenging but most exciting task, leading IDSs toward artificial intelligence.

Bibliography

[1] 'Intrusion Detection, network security beyond the firewall', by Terry Escamilla, ISBN 0471290009

[2] 'Intrusion Detection', by Rebecca Gurley Bace, ISBN 1578701856

[3] 'Security in computing', by C.P. Pfleeger, ISBN 0131857940.

[4] 'Hacking exposed, 2nd edition', by J. Scambray, S. McClure, G. Kurtz, ISBN 0072127481

[5] 'TCP/IP Illustrated vol.1', by R.W. Stevens, ISBN 0201633469

[6] 'Les réseaux', by Guy Poujeole, ISBN 2212089678

[7] 'Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection', by T.H. Ptacek & T.N. Newsham

[8] 'A strict anomaly detection model for IDS', Sasha & Beetle, Phrack 38

[9] 'From virus to antivirus', by Marc Ludwig